

Undergraduate Thesis

Major in Mathematics

Faculty of Mathematics
University of Barcelona

Electrical Distribution Networks: The Power Flow Problem

Author: M. Pilar Pujol Closa

Advisor: Dr. Antoni Benseny

Department: Matemàtica Aplicada i Anàlisi

Barcelona, July 19, 2016

Abstract

In this project a method to solve the Power Flow Problem for Electrical Distribution Networks was developed. Electrical Networks are studied using electrical graphs and hierarchical algorithms are implemented to traverse them. The proposed solution employs numerical methods using C++ to solve the power flow problem and profits from the hierarchical structures created, which makes the solution more efficient. Additionally, a simulator was developed in order to visualize the Electrical Distribution Network and displays results in a more intuitive manner for any specific configuration of the power flow problem. This visualizer also allows for easy interaction with the program. Operating conditions are varied to test the accuracy and useful range of predictive solutions to the power flow problem in the methods described, including areas for future improvement.

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Benseny. Without his guidance and assistance this project would have never been accomplished. I would also like to thank him for all the help and support provided during my years as student. I am also grateful to my family and friends for their support not only during the development of this project but also during the years I have been studying.

Contents

Introduction	1
1 Power Flow in Distribution Networks	3
1.1 The Electrical Energy Systems	3
1.2 Description of the Problem	4
1.2.1 Electrical Information	4
1.3 The Power Flow Problem	5
1.3.1 The Basic Nodal Method with current approach	6
1.3.2 The Basic Nodal Method with power approach	7
1.3.3 Solving the Power Flow Problem for radial networks	8
1.3.4 Solving the power flow problem for weakly meshed Networks . . .	10
2 Graph Data and Algorithms	13
2.1 Data format for graphs	13
2.2 A brief review on hierarchical traversal of graphs	13
2.2.1 Breadth First Search (BFS)	14
2.2.2 Depth First Search (DFS)	15
2.3 Data structure for meshed trees	17
3 Electrical data and algorithms implementation	21
3.1 Reading the electrical Data	21
3.2 Computation of double sweep iterations	21
3.3 Computation of Power compensations	22
3.4 Losses in power of the system	24
4 Electrical Distribution Network Simulator	25
5 Results	27
Conclusions	31

Introduction

Since the use of electrical energy became extended to most populations, the electrical networks have not stopped growing. In order to become more efficient, deep studies in this networks were required. One of the most common problems encountered in this type of analysis is the Power Flow problem. An appropriate mathematical approach can provide rigorous insight and a more consistent solution. I am a student of the double major of Physics and Mathematics offered in this University and I thought it would be interesting to approach a complex problem that combines knowledge of both of majors. I wanted to study a project involving networks since Graph Theory is one of the courses I enjoyed most. In this project I make use of skills in programming, numerical methods, and graph theory which I have acquired during my studies.

An iterative program implemented in *C++* which analyzes and solves network problems efficiently was developed. The *C++* platform allows for the inclusion of libraries such as OpenGL and FLTK, both of which were used.

The main goal of this project is to mathematically model sample Electrical Distribution Network by means of graphs and complex electrical variables. In order to achieve this, it is necessary to structure the connectivity of the networks through meshed networks, analyze, and solve the power flow problem in this network by iterative numerical methods. This project is composed of six chapters. In the first one electrical networks are introduced.

The power flow problem is described and some approaches for different kinds of systems are given. Finally, a solution method for Distribution Electrical Networks is outlined. In the second chapter we explain how the information that relates to the structure of the network is treated. The way we treat the structure data is crucial to effectively find the solution of the problem. We have used our knowledge in graph algorithms to do this. In the third chapter we explain how the electrical data are treated, and how the solution of the problem is computed by using numerical methods. In the fourth chapter we describe the creation of a platform that allows interaction with the program and shows on screen the results. Finally, the results are discussed in chapter five and some conclusions are given in chapter six.

Chapter 1

Power Flow in Distribution Networks

1.1 The Electrical Energy Systems

At the very beginning of the usage of electrical networks, power plants used to directly supply the population. Since then, a lot of improvements have been made in order to increase the quality of the provided service and the efficiency of the networks while these were growing to supply an increasing number of customers. Nowadays the transportation of the electrical energy takes place in super-connected lines, with a lot of nodes and connections between them. This structure is preferred in order to reduce the affectation to a maximum number of customers in the case that something in the system would not working properly. For example, if a power plant reduces its production for technical reasons or if one line gets damaged for external conditions such as rain this structure aims to avoid a population from not having service or to reduce the period of time without service. As a result there are many different power plants connected to many customers at the same time. This allows to reach a steady state aside of small variations such as failure in one of the lines connections, changes in the power generation or in the power consumption, etc. The electrical energy systems are structured in three different levels: generation, transportation, and distribution. In between the different levels voltage transformation centers and substations may be found [5]. This are the different levels in which the electrical networks are structured:

- **Power Plants:** Is where the electrical energy is "obtained/transformed" from different types: hydraulic, thermic, nuclear thermic... The power generators usually provide energy with a voltage between 6 and 25 kV.
- **Power Transformer Parks:** In order to diminish the power losses by Joule's effect, the voltage of the electrical energy coming from the power plants has to be increase in power transformers.
- **High Voltage Lines:** After going through the transformer parks the electricity is then distributed through the High Voltage Lines. In Spain the stablished voltages that this lines must conduct the current at is stated by law to be of 132, 220, and 380 kV.

- **Primary Distribution Networks:** These lines conduct the current with lower voltages than the High Voltage Lines, usually around 45, 66 or 132 kV. Therefore, the voltage is changed again in between the transportation lines and the distribution lines, in power transformer substations. Primary distribution networks are highly meshed and usually have a length smaller than 100 km. Typically these lines surround cities.
- **Medium voltage distribution networks** are usually shorter than 25 km and their voltage is established to be of 3, 6, 10, and 20 kV. Hence, the voltage is reduced again in what are called distribution substations. If the lines supply rural areas and industrial consumers then they are usually land lines. However in the cities these lines are found underground, and in the gravity center of consumption areas the voltage is again changed in transformation centers.

The networks that supply villages usually have a radial structure. Whereas in the cities, in order to guarantee the power supply, the distribution networks are usually meshed. However they are provided with lots of switches allowing them to work as weakly meshed radial networks. If one of the lines gets damaged, just by changing the position of some of the switches the consumers will still have power. Allowing the line to be fixed while still providing consumers with power. Nevertheless it is unavoidable to have a temporary stop in the power supply to the consumers that are primarily supplied by that line.

- **Low voltage distribution networks:** usually have a length of the order of 1 km and are the ones that provide service to most consumers, houses, industries, and companies that use low voltage. The standardized voltage values for these networks in Spain are of 230 V and 400 V. The structure of these networks is usually radial or meshed with switches that work as radial like on the medium voltage networks.

1.2 Description of the Problem

Electrical Distribution Networks can be mathematically represented by electrical graphs. An electrical graph is a graph that includes electrical information. The vertices will be those nodes where there is generation, consumption or transformation of electrical energy. The vertices are connected by conductive materials, called branches, which are characterized by their impedance.

1.2.1 Electrical Information

The impedance Z indicates the opposition of a branch to alternating current when voltage is applied.

$$Z = R + iX, \quad (1.2.1)$$

where R is the resistance and X is the reactance. The inverse of the impedance is the admittance which is the measure of how easily the branch allows current to flow.

$$Y = \frac{1}{Z} = G + iB, \quad (1.2.2)$$

where G is the conductance and B the susceptance. Additionally the current modulus may be found by:

$$J = \frac{V}{|Z|}, \quad (1.2.3)$$

where J stands for the magnitude of the current, V the magnitude of the voltage, and $|Z|$ the magnitude of the impedance. The goal is to find the relationship between the injected currents and the voltage in the vertices of the graph. The injected currents will be positive if that vertex "generates" electrical energy or negative if the vertex is a consumption vertex. One may see the current flow problem described in terms of admittances instead of impedances.

Let us consider two nodes j, k with complex voltages E_j and E_k respectively. Let us consider a branch that links node k with node j and has an admittance y_{kj} . By using Ohm's law we can find the relationship between the current on the branch and the voltage difference in the considered nodes:

$$I_{kj} = y_{kj} (E_k - E_j). \quad (1.2.4)$$

And by charge conservation, the injected current in node k must be equal to the sum of the current of the branches that "leave" the vertex:

$$I_k = \sum_{j=0}^n I_{kj} = \sum_{j=0}^n y_{kj} (E_k - E_j). \quad (1.2.5)$$

If we write this for every node, we have a system of n equations.

The Current Flow Problem aims to find the nodal voltages E_j for every node j and the current flow I_{kj} in the branches that connect the nodes j and k that are consistent with the known nodal currents I_k in the k nodes.

1.3 The Power Flow Problem

The power flow problem is the perfect tool to study a transport system or an Electrical Distribution network. Additionally Electrical companies have information about power consumption instead of the injected currents, the power flow problem may be of a bigger interest. We know the power that consumers generally take and the power that power plants supply, so this allows us to find the voltages in each node and the active and reactive power in every system (line, transformers, capacitors...). The problem will be nonlinear due to the power and voltage constraints. Hence, the numerical solution will be iterative.

On the formulation of the power flow problem there are some assumptions that have been made from the real system. We assume that the steady-state is achieved and the system

has constant frequency and constant voltages. Furthermore, the nodes will be classified and sometimes treated differently according to the information each of them provide.

- 1 Voltage controlled node or PV node:** The total injected active power P_i is known and the voltage magnitude V_k is steady to a specific value due to reactive power injection. Mechanically one may adjust this two values to the desired ones by changing voltage regulators for example.
- 2 Nonvoltage controlled node or PQ node:** The active and reactive power are the known variables in this node. Also there is no generation in this node or it is taken as a constant and used as a known value. This node would usually correspond to a consumer.
- 3 Slack node:** If all the nodes were either PV or PQ, we would already know all the power values P_i which is not compatible with the active power balance that we have. The losses mainly due to Joule's effect of the system are not precisely known in advance of the load-flow calculation. The total injected power cannot be specified at every single node. Usually one of the voltage-controlled nodes is used as a slack node with unspecified active power. The slack node is usually chosen to be one of the most important generators in the system or the linking node of the system we are studying to the rest of the total system.

In our project we will be studying **Electrical Distribution Networks**, that have the particularity that are weakly meshed networks. Therefore, our results will not be able to be applied to networks that are highly connected for example.

1.3.1 The Basic Nodal Method with current approach

In the Basic Nodal Method the variables are the complex node voltages and currents. Usually the voltage magnitude reference will be the ground, and the voltage of one of the nodes will be taken as zero (usually the node that we will designate as a slack node). The nodal current will be the net current injected into the network at a given node from the outside. The current may be positive if it is entering the network or negative if it is leaving the network. The net nodal injected current will be the algebraic sum of each nodal current.

As in eq. (1.2.1) using E_j to designate the voltage of the node j and using y_{jk} for the admittance of the branch between nodes j and k . The current in this branch will be given by:

$$I_{kj} = y_{kj}(E_k - E_j). \quad (1.3.1)$$

Now, if we numerate from 0 to n all the nodes where denotes 0 the reference node or the ground. Kirchhoff's current law states that the injected current I_k in node k must be equal to the sum of the currents leaving the node.

$$I_k = \sum_{j=0}^n I_{kj} = \sum_{j=0}^n y_{kj}(E_k - E_j). \quad (1.3.2)$$

Writing this for every node, and since the ground voltage is zero $E_0 = 0$, we have a linear system,

$$I_k = \sum_{j=0 \neq k}^n y_{kj} E_k - \sum_{j=1 \neq k}^n y_{kj} E_j. \quad (1.3.3)$$

Writing this for every node, we have a system of nodal equations:

$$I_k = \sum_{j=0}^n Y_{kj} E_j \quad k = 0, \dots, n, \quad (1.3.4)$$

where

$$Y_{kk} = \sum_{j=0, j \neq k}^n y_{kj}, \quad Y_{jk} = -y_{kj} (k \neq j). \quad (1.3.5)$$

The diagonal terms Y_{kk} are found by adding the admittances that are parallel to the ground and the admittances that are on series of the incident branches in every k node that connect their adjacent nodes. The non-diagonal terms Y_{kj} correspond to the negative value of the admittances of the different pair of nodes k, j when are connected by a branch, if they are not connected this value is zero. Y is a complex symmetric matrix of squared order $n \times n$ known as the admittance matrix which allows modeling the network and contains electrical information of the branches.

1.3.2 The Basic Nodal Method with power approach

As mentioned, we are trying to solve the Power Flow Problem and not the Current Flow Problem since the latter has already been extensively studied. Analogously to the last section we will now develop the Basic Nodal Method but with nodal injected powers. We will translate the current flow problem into the power flow problem just by using some formulas. However this time at the end of the derivation we will have a set of nonlinear equations.

The power flow problem consists of finding the nodal voltages for every vertex and the nodal currents for all the edges under the assumption that the system has reached the steady state. The known parameters will be the power consumption, the power generation, and the impedances of the branches. The unknowns are the power lost, the current flow through the branches, and the voltages at most of the vertices.

From the known voltages the active and reactive power flow values will be found, as well as the losses of the system. To solve the problem we will apply Kirchoff's laws as mentioned and charge conservation. Thus:

$$S_j = E_j I_j^* = E_j \left(\sum_{k=1}^n Y_{jk} E_k \right)^* \quad j = 1, \dots, n, \quad (1.3.6)$$

where

$$E_j = V_j e^{i\delta_j}, \quad Y_{jk} = |Y_{jk}| e^{i\gamma_{jk}}, \quad \delta_{jk} = \delta_j - \delta_k. \quad (1.3.7)$$

We have a non linear system of n complex equations, which we can break into a set of $2n$ real equations:

$$P_j = V_j \sum_{k=1}^n V_k |Y_{jk}| \cos(\delta_{jk} - \gamma_{jk}) \quad j = 1, \dots, n \quad (1.3.8)$$

$$Q_j = V_j \sum_{k=1}^n V_k |Y_{jk}| \sin(\delta_{jk} - \gamma_{jk}) \quad j = 1, \dots, n. \quad (1.3.9)$$

In each node we have 2 associated equations and 4 electrical variables, which not all of them are not known for each node.

- P_j - Active power
- Q_j - Reactive power
- V_j - Voltage magnitude
- δ_j - Voltage phase angle

Over the increased usage of networks there have been developed many different methods to approach the load flow problem such as The Gauss-Seidel method, the Newton-Raphson method, and the Fast Decoupled Load Flow. While these techniques are able to provide a very efficient and accurate solution for transportation electrical networks, they may become inefficient for weakly meshed networks. The reason for this is that these techniques might encounter difficulties with poorly initialized networks or with networks that have a substantially different structure.

1.3.3 Solving the Power Flow Problem for radial networks

For Electrical Distribution Networks we will also apply the Basic Nodal Method but with some modifications. Electrical Distribution Networks are weakly meshed networks that can be treated almost as radial networks. We will explain first what the solution for a radial network would consist of, and later what the solution for the weakly meshed network will be. The radial electrical networks will be modeled with electrical trees. These graphs will be traversed by using algorithms that will be explained in the next chapter. This algorithms will provide us with a list of vertices and for every vertex k there will

be another vertex j that will be considered the parent of k and the edge that links them will be designated as predecessor edge k . A hierarchic language will be extensively used when referring to traverse the network. Ohm's law relates the voltage drop in every pair of adjacent vertices with the current \mathcal{I}_k of the linking edge that has an impedance value of z_k :

$$E_j - E_k = z_k \mathcal{I}_k, \quad (1.3.10)$$

The notation we use is as in figure 1.3.3. Additionally, by charge conservation in vertex j we know that the arriving current \mathcal{I}_j to j through its predecessor edge j will be equal to the sum of the current arriving to their descendants k plus the injected current I_j in vertex j

$$\mathcal{I}_j = \sum_{k \text{ descendant of } j} \mathcal{I}_k + I_j. \quad (1.3.11)$$

We know that $S_k = E_k \mathcal{I}_k^*$. Therefore, if we multiply the conjugate of this last expression by the voltage E_j of vertex j we have the following power relationship:

$$E_j \mathcal{I}_j^* = \sum_{k \text{ descendant of } j} E_j \mathcal{I}_k^* + E_j I_j^* \rightarrow \mathbf{S}_j = \sum_{k \text{ descendant of } j} \tilde{S}_k + S_j. \quad (1.3.12)$$

Thus, the accumulated power \mathbf{S}_j is the sum of the sent powers \tilde{S}_k to their descendants k plus the injected power S_j to j . Using Ohm's law, we know that the send power to the node k will be the loss power in the predecessor branch and the accumulated power S_k in the k node:

$$\tilde{S}_k = z_k \mathcal{I}_k \mathcal{I}_k^* + E_k I_k^* = z_k \mathcal{I}_k \mathcal{I}_k^* + S_k. \quad (1.3.13)$$

To solve the flow problem an iterative method is used, in the first iteration the voltage of all the nodes is equal to the voltage of the slack node (the root of the tree). Iteratively this values are corrected until the value found on the new iteration differs from the last one by less than a given tolerance. The iterations are done as follows: Starting at the consumption vertices we can traverse the graph upwards, new values for the accumulated powers are found, and the total power at the root of the graph is found. In the k node, the accumulated powers are the injected powers and this is at its time also accumulated to its parent j the send power:

$$\mathbf{S}_{k+} = S_k, \quad \mathbf{S}_{j+} = z_k J_k^2 + \mathbf{S}_k \quad \text{where} \quad J_k^2 = \frac{\mathbf{S}_k^* \mathbf{S}_k}{E_k^* E_k}. \quad (1.3.14)$$

In the downwards sweep the nodal voltages are found, the voltage in vertex k is found from the voltage value of its parent j :

$$E_k = E_j - z_k \frac{\tilde{S}_k^*}{E_j^*}. \quad (1.3.15)$$

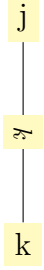


Figure 1.3.1: Vertex parent j, linked to vertex k by the edge k.

At the first iteration all the nodes have the same voltage, the voltage of the slack node. New values for the voltages are found recursively. The iteration stops when the new found values differ from the ones of the last iteration less than the value of a prestablished tolerance. This iterative method that uses the hierarchic structure of distribution networks is known as the Shirmohammadi-Hong method.

1.3.4 Solving the power flow problem for weakly meshed Networks

For Radial Electrical Networks we could apply the Shirmohammadi-Hong method ref. [8] . However, as it has been previously mentioned, Electrical Distribution Networks are weakly meshed networks and the solution of the problem is constructed in a similar but slightly different manner.

Weakly meshed graphs will be transformed into spanning trees by the following procedure. In order to obtain a radial network and then solve the power flow problem the cycles of the meshed network will have to be broken. One of the edges that closes the cycle will be removed. However, a new additional vertex that we will call *alias vertex* will appear linked by a new edge to one of the vertices that the removed edge was linking. The *alias vertex* will have the same electrical properties as the other vertex that the broken edge was linked to, the one that would be substituting. This other vertex will be referred to as *original vertex*. And the concept of *alias pairs* of vertices will be used when referring to an *alias vertex* and its *original vertex*. By doing this we will build a spanning tree, which will not have cycles. Likewise in the radial case, upwards and downwards sweeps will be performed in order to find the power and voltage in every node. The only difference is that now further calculations will need to be done since the voltage in a vertex and its alias has to be the same and probably that will not happen straight forward after the first calculation.

To compensate for breaking the loops it will be required to compute compensating (active and reactive) power injections in both vertices of the alias pairs in order to simulate the

current flow from the original loop. The injection of power will be positive or incoming in one vertex and negative or outgoing in the other one (its alias).

For this purpose the sensitivity matrix needs to be introduced. We will give a first notion about what it is and why it is important, this matrix describes the approximate affectation to the voltage mismatches at all alias pairs due to the positive/negative injection compensations of power at every the alias pair. Therefore, the sensitivity matrix gives information about how the voltage discrepancies of all alias pairs are affected by unitary positive/negative power injections at all alias pairs; that is, around their associated cycles. An approximation for these unitary power injections will be done by means of unitary currents injections.

The reactive compensation in PV nodes will be treated in a similar manner by an adaptation of the sensitivity matrix.

Once the sensitivity matrix M is computed, a linear system has to be solved in order to compute the power injection corrections ΔE at all alias pairs and PV nodes associated to the voltage discrepancies ΔE at all the alias pairs and PV nodes:

$$M\Delta S = \Delta E \quad (1.3.16)$$

Summarizing, in order to iteratively correct the voltage mismatches at alias pairs and PV nodes a compensation method is used until they are lower than a previously established tolerance.

Each iteration has three steps:

- **Step 1** Voltage mismatches in alias pairs and PV nodes are calculated solving the radial power flow problem for the consumed and compensating power injection known.
- **Step 2** Power injection compensating corrections are computed by solving a linear system with the sensitivity matrix to approximately vanish the voltage mismatches.
- **Step 3** Correction of the compensating power injections by adding the found correction to the previous compensating power injections.

Chapter 2

Graph Data and Algorithms

In this chapter we will explain how the structural information of the network is introduced by means of graphs. Some methods to traverse graphs will be reviewed, and how the graph information is treated and prepared for being used by the algorithms described in the following chapter. The construction of structures as spanning trees and efficient traversal methods to access the information is crucial to solve the power flow problem.

2.1 Data format for graphs

The program reads the data of the electrical network from a "data.dat" file where the main information of the considered system is found: the number of vertices with their associated power and voltage information, information of the edges such as which two vertices are linked by them and their impedance information, there is also information about the factor of power consumption of the customers. All this read information is coded and structured in the program and printed again in a "data.out" file. We will now explain how the information that refers to the structure of the network is processed and will leave the treatment of the electrical information for future sections.

The first number that will be read from the "data.dat" file is the number of vertices v_n of the graph which will be followed by the number of edges e_n . Then we will find v_n rows with electrical information for each vertex. After this, we will have e_n rows with information about the edges. In the first column the number of the edge will be given, the second column and third column will describe which vertices are linked by the given edge.

This information is used to obtain the adjacency lists for all vertices. The adjacency list of a vertex contain all its neighbors or adjacent vertices.

2.2 A brief review on hierarchical traversal of graphs

We have mentioned that how the graphs are traversed will be crucial for solving the problem. There are many different ways to traverse a graph but for the purposes of our project, a hierarchical algorithm will be the best way to traverse a graph. It will provide us with an ordered list of vertices that will be used later to perform calculations and to find the solution of the Power Flow Problem of a given network.

2.2.1 Breadth First Search (BFS)

The figure consists of four diagrams arranged in a 2x2 grid, each showing a binary tree structure. The nodes are labeled with numbers 0 through 9. The root node is always 0. The edges are labeled with 0 or 1. The nodes are colored blue (internal nodes) or light blue (leaf nodes). The sequence of diagrams illustrates the steps of a Huffman tree construction algorithm:

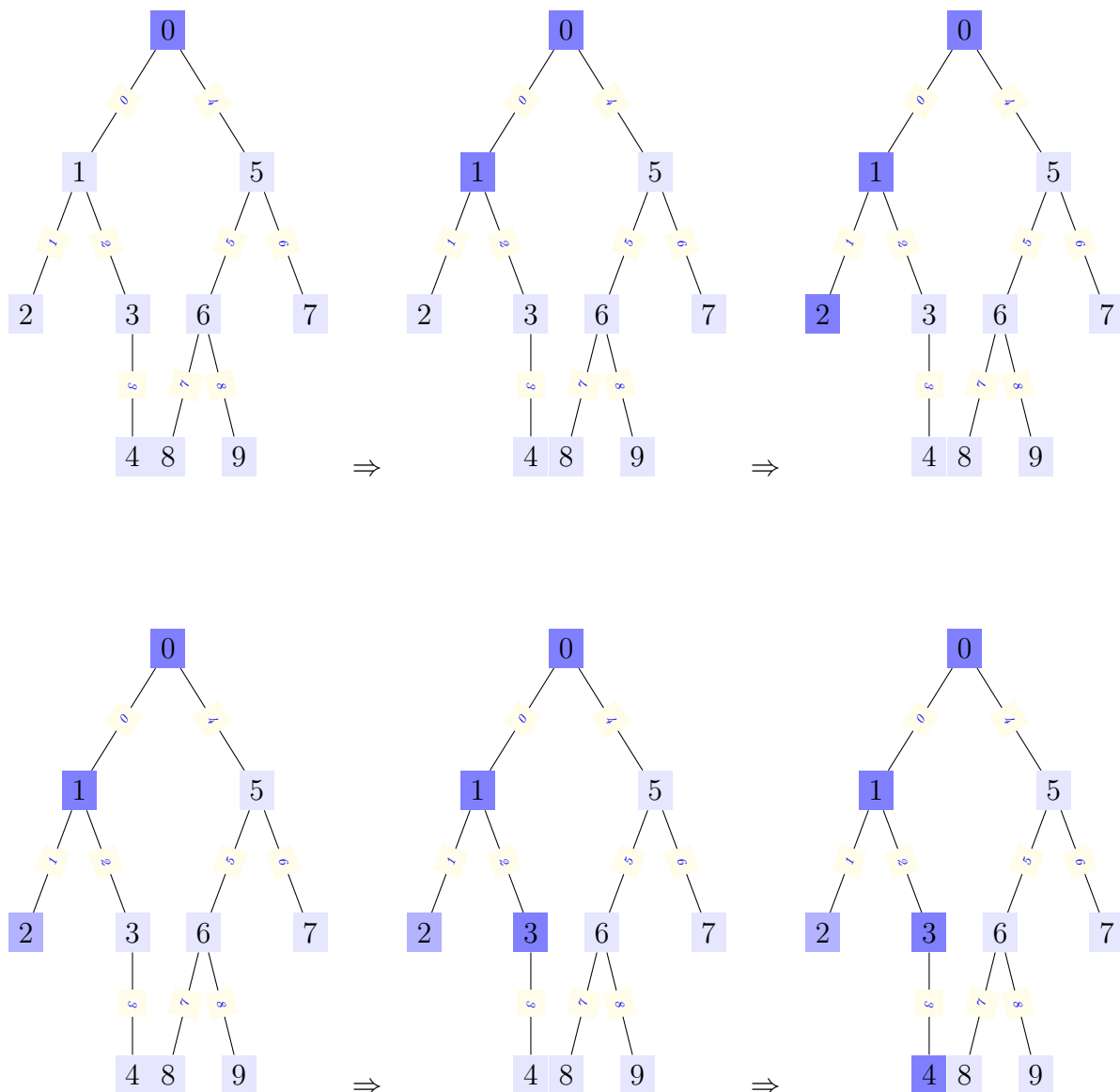
- Diagram 1 (Top Left):** The root node 0 is blue. It has two children: 1 (left, blue) and 5 (right, blue). Node 1 has children 2 (left, light blue) and 3 (right, light blue). Node 5 has children 6 (left, light blue) and 7 (right, light blue). Node 3 has child 4 (light blue). Node 6 has children 8 (left, light blue) and 9 (right, light blue). All edges are labeled with 0 or 1.
- Diagram 2 (Top Right):** The root node 0 is blue. It has two children: 1 (left, blue) and 5 (right, blue). Node 1 has children 2 (left, light blue) and 3 (right, light blue). Node 5 has children 6 (left, light blue) and 7 (right, light blue). Node 3 has child 4 (light blue). Node 6 has children 8 (left, light blue) and 9 (right, light blue). All edges are labeled with 0 or 1.
- Diagram 3 (Bottom Left):** The root node 0 is blue. It has two children: 1 (left, blue) and 5 (right, blue). Node 1 has children 2 (left, light blue) and 3 (right, light blue). Node 5 has children 6 (left, light blue) and 7 (right, light blue). Node 3 has child 4 (light blue). Node 6 has children 8 (left, light blue) and 9 (right, light blue). All edges are labeled with 0 or 1.
- Diagram 4 (Bottom Right):** The root node 0 is blue. It has two children: 1 (left, blue) and 5 (right, blue). Node 1 has children 2 (left, light blue) and 3 (right, light blue). Node 5 has children 6 (left, light blue) and 7 (right, light blue). Node 3 has child 4 (light blue). Node 6 has children 8 (left, light blue) and 9 (right, light blue). All edges are labeled with 0 or 1.

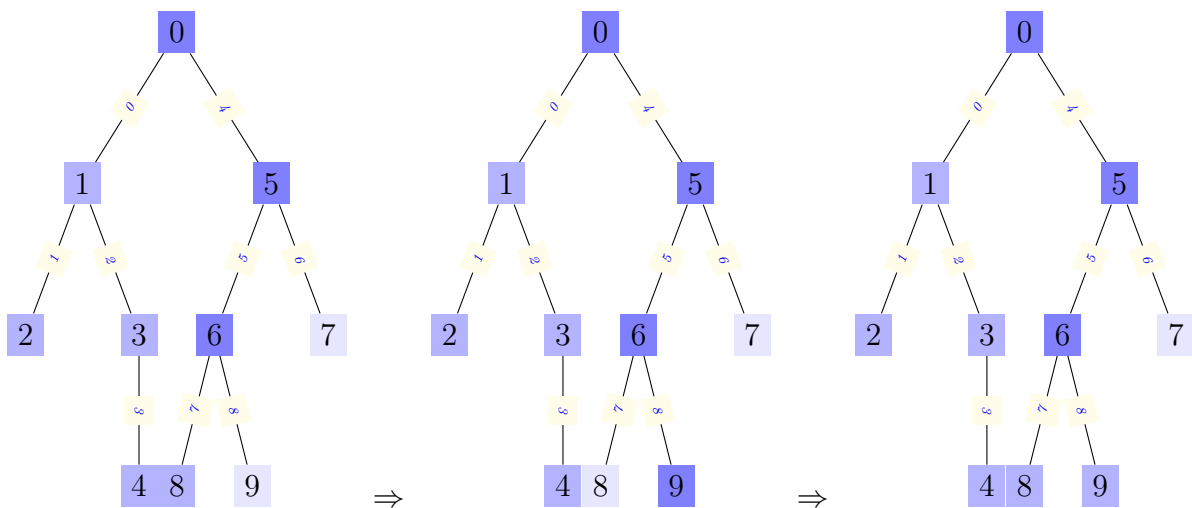
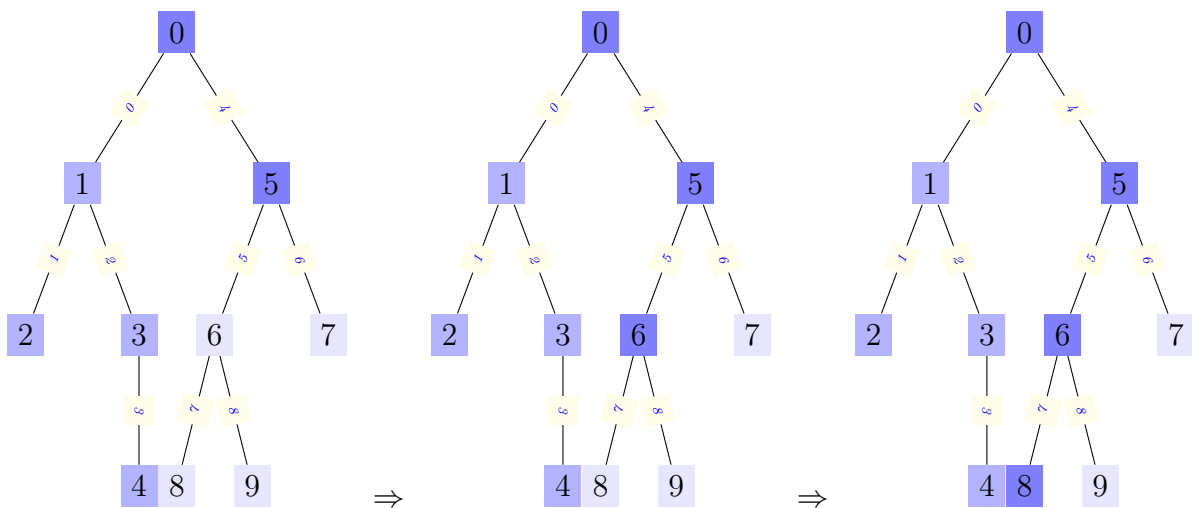
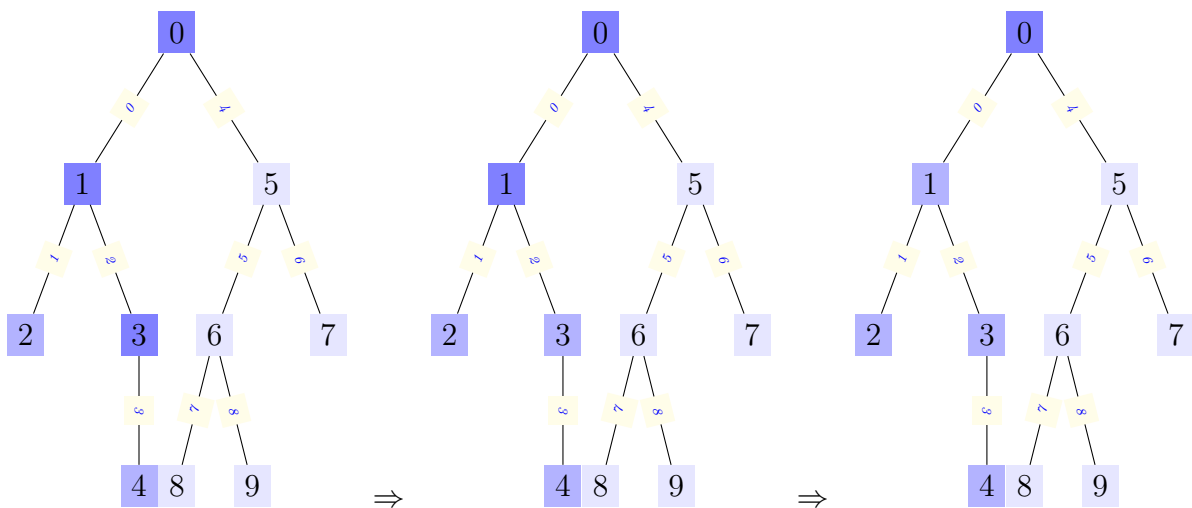
To avoid making this algorithm work for an infinite amount of time we need to keep a record of the visited nodes for example by saving them into an array. If all the edges weight the same this algorithm provides us with the shortest path from the root to all the other vertices. It also informs us about the connectivity and the number of components that the graph has.

2.2.2 Depth First Search (DFS)

This other algorithm traverses the graph in a more recursive way. As one may infer from the name it goes to the deepest level from neighbor to neighbor before backtracking. It does not visit any vertex twice either.

This example will help us understand it better, the legend of colors is the same as used before.





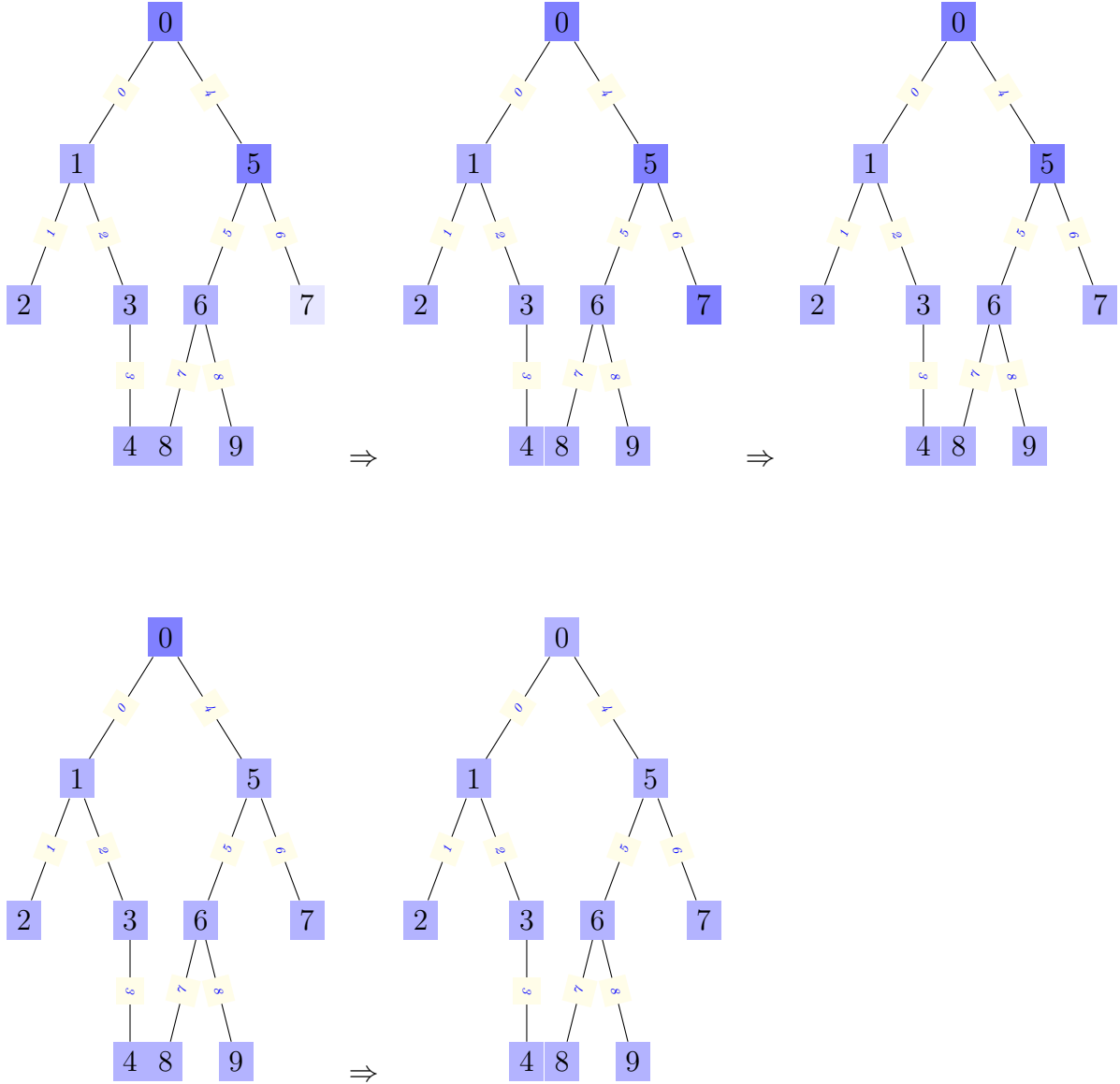


Figure 2.2.2: The different steps of how the vertices are traversed with the DFS algorithm is shown in this figure.

2.3 Data structure for meshed trees

The program implements the DFS and the BFS algorithms obtaining a hierarchic list of vertices. The main difference between the BFS and the BFS Meshed tree, and between the DFS and the DFS Meshed tree is basically the same. We have already explained the difference between the BFS and the DFS in a previous section. Here we will explain how we have modified the algorithms in order to obtain the desired spanning trees.

First of all, every time that the program finds a root it builds a new tree. The information of the vertices is saved by using six main vectors that list:

- the ordered vertices
- the indices of vertices
- the upper vertex of every vertex
- the upper edge of every vertex
- the depth of every vertex
- the vertices of whom are alias (itself or original vertex for alias ones)

One may notice that the spanning tree will have more vertices than the original graph, but the same number of edges. The number of vertices of the new tree will be the number of edges +1. And thus, the dimension of all the vectors that will contain information of the vertices will be the number of edges +1. The alias vertices will be saved in the vertices vector starting at the position v_n the number of vertices. Hence from 0 to the number of vertices, it will all be *original vertices*; and from there to the number of edges +1 all the vertices will be alias vertices.

The program starts by finding a root, and saving its data: depth will be equal to 0, the parent of a root vertex will be itself since it has no parent, and the alias will be again itself. Once we have found a root we look for unvisited adjacent vertices, following the BFS or the DFS algorithm accordingly to which tree we are constructing. We will treat each vertex differently according to whether is an alias or not:

- If a vertex is **not an alias** vertex: we save it in the vertices vector among the first positions, in the alias vector we save itself as its alias, we also save the vertex from we have found it is adjacent to as its parent, and its depth as the parents depth +1.
- If it is an **alias**: the original vertex will be saved as its alias in the alias vector (instead of itself), and finally the parent will not be the parent from the original vertex nor the original vertex, it will be the other vertex that the edge links the original vertex to.

However the edge that links the original vertex to the parent of the alias will also be saved. In the "*data.out*" file that the program is writing it will appear a message description of this edges saying that they are *out of tree*, since it will not appear on the spanning tree that we are building. The tree is usually traversed upwards, every edge goes from a vertex to its parent, and when the parent is the same vertex we are considering, the program knows it has found a root and it stops.

Here we provide an example of how a meshed graph is transformed into a spanning tree. The number of the vertex includes information about how the graph is traversed by using the BFS method or the DFS method respectively:

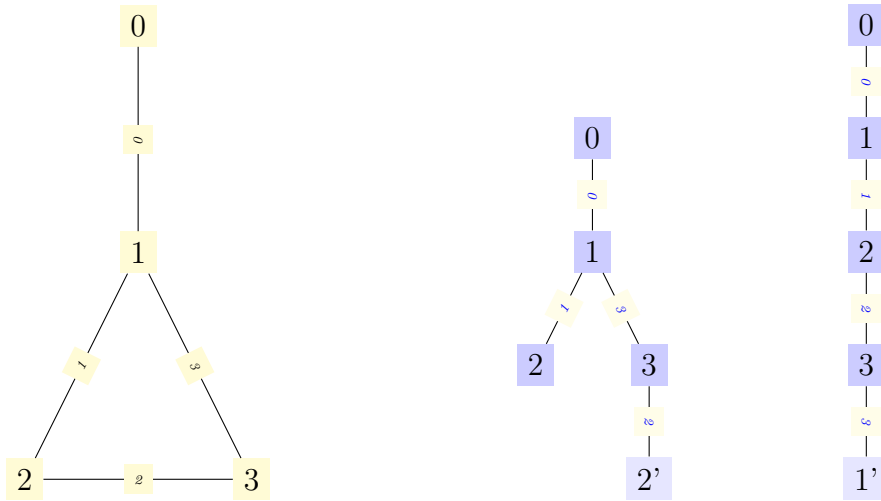


Figure 2.3.1: At the left of this figure there is the tree before any modification, in the middle its BFS tree and at the right its DFS tree.

Example 2.3.1.

The program will write the following information of the vertices: vertex, its alias, number of vertex, depth, and its parent it does not write the two last things for a root vertex. Then it writes all the edges by going through the alias vertices vector. It writes from an alias vertex to its parent, and according to if its alias is itself or not it will additionally print a message saying edge in tree or out of tree. It also writes the paths to root from each of the vertices in the spanning tree. Again, it is able to distinguish if a vertex is an alias or not by knowing who is saved in the alias vertices vector, when the parent of a vertex is itself the program stops knowing that it has found the root.

Finally it is able to find and describe the different cycles that the original graph has. To do so, the program starts by visiting the vertices of the vertices vector by starting at the first alias position. If its depth is different from the original vertex depth, it visits the alias parent. If this still has a different depth than the original vertex it keeps backtracking among the alias predecessors until finds the one with the same depth. Then if the parent of this new found vertex is not the same as the parent of the original vertex it does backtracking for both branches until it finds the "first common ancestor".

Very similarly in order to write the correct BFS cycles that it has found, starts by visiting every alias vertex. The difference now with the previous algorithm described is that we will count every visited vertex when backtracking and will construct a vector of the dimension of this number. In this vector the first vertex saved will be the alias vertex we are visiting, which will be followed by their predecessors that the program will visit when backtracking to equal the depth to the original vertex depth. Once we are in equal depth if the parent of this vertex is still not the same as the original vertex parent we will backtrack through both branches until the first common ancestor is found. Each of the backtracked vertices will be saved in the created vector. The predecessors of the alias vertex will keep being saved following the previously saved. The original vertex will be the last vertex in the vector, and its ancestors will be written in the previous positions. The first common ancestor found, will then be saved in between the backtracked vertices of both branches, completing the cycle. Then the cycles can be easily written just by writing this new vector.

In figure 2.3 we can see an example with the initial graph and the spanning tree that the program builds. To break the cycles, two edges have been broken, the 7 and the 8. We have two alias vertices: 6', and 5'.

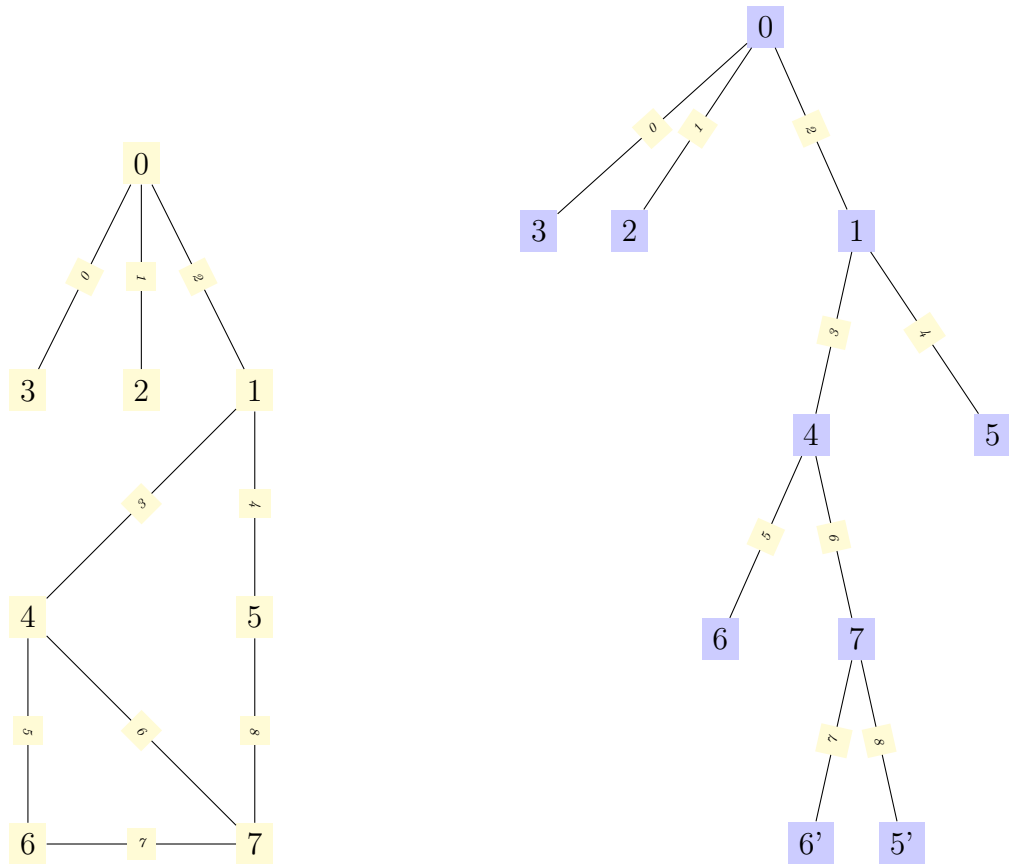


Figure 2.3.2: The original tree on the left with the BFS spanning tree at the right.

Considering the alias pair 6'-6, the alias vertex is 6' which has a depth equal to 4, and the original vertex 6 has a depth equal to 3. To find its corresponding cycle:

- backtracking edge number 7 (from 6' to 7) equals depths to 3,
- backtracking edges number 6 (from 7 to 4) and number 5 (from 6 to 4) we arrive to the first common ancestor vertex number 4,
- the found cycle is 6'-7-4-6.

In the alias pair 5'-5, the alias vertex is 5' at depth 4 and the original vertex 5 is at depth 2. To find the corresponding cycle:

- the edges 8 (5'-7) and 6(7-4) are backtracked equaling the depth to 2,
- the edges 3(4-1) and 4(5-1) are also backtracked and the first common ancestor 1 is reached,
- the found cycle is 5'-7-4-1-5.

Chapter 3

Electrical data and algorithms implementation

In this chapter we will explain how the electrical information is treated. How the sensitivity matrix is calculated, and how the different parts of the program interact between each other and build the solution of the problem described in section 1.3.4.

3.1 Reading the electrical Data

As mentioned previously the information is read from a file that contains the data. Additionally to the number of the number of vertices v_n and the number of edges e_n , in the first row we can read the number of PV vertices. Then we will have v_n rows that will contain information of every vertex starting in 0 and ending at $v_n - 1$; followed by its positions px , py and its nominal active and reactive powers, and finished by a number that will indicate whether or not it is a PV node. If so, an additional number will follow this one: its consigned voltage magnitude. Then in the following rows, after the first three columns that contain the number of the edge and the two vertices that links, there are two more columns, with the resistance and the reactance of the edges. It is important to point out that in the data.dat file the unitary values for voltages are 25 kV, for impedances 625 Ω , and for current 400 A. On the last row of the data there is the consumption factor μ that has a value between zero and one. This value gives information of the consumed power. The power consumption $S = S_c$ that a consumption node has can be calculated by $S_c \simeq \mu S_p$ where μ is the factor consumption and S_p is the purchased power. The electrical information such as impedances, currents, and powers are structured in trees. The voltages are coded in a vector with two components for each vertex.

3.2 Computation of double sweep iterations

In this section we focus on explaining how the computation of the explained solution of section 1.3.3 is done. With the same notation as used before we explain how the double sweep is performed. In the upwards sweep just by following the vertices vector we obtain the vertices in a hierarchical order. This provides us with the k vertex, and the upper vertices vector tells us its predecessor j and the upper edge tells us its predecessor edge k .

In the downwards sweep, we know all the information of the upper vertex j , and we can obtain the new value for the voltage of the vertex k .

3.3 Computation of Power compensations

The computation of the power compensations will profit from the hierarchic structures that we have. Let us consider a weakly meshed network with at least two cycles, named vc, uc . We will have two corresponding alias pairs $v - av$ and $u - au$. We will study how the voltage discrepancy in the alias pair $u - au$ is affected by a unitary current in the other cycle with alias pair $v - av$. Due to the fact that nodal voltages are similar, power injections can be used to approximate current injections $\Delta S_{vc} \approx \Delta I_{vc}$. We will explain now how a unitary power injection will be done in the cycle $v - av$:

First of all, the program will backtrack from the alias vertex until it finds a vertex at the same depth as its original vertex. When doing this, a value of $J = -1$ will be assigned to each of the backtracked edges. Once we have found a predecessor of the alias vertex at the same depth as the original vertex, the program will backtrack through both branches until the first common ancestor is found. While backtracking every visited edge will have an assigned value for J , if the vertex is a predecessor of the alias vertex J will have a value of -1 but if it is a predecessor of the original vertex it will then have a value of 1 . After this procedure a downwards sweep will be performed, and by using Ohm's law we will be able to find the voltage drop in all vertices due to a unitary current in the cycle $v - av$. The voltage drop will be the accumulated impedance with positive or negative sign. If z_u is used to described the impedance of a vertex, and z_{uu} the impedance of its parent and z_{ue} the impedance of the edge:

$$z_u = z_{uu} + Jz_{ue}. \quad (3.3.1)$$

We will apply the explained method in example 3.3.1. Taking the cycle $5 - 5'$ as the cycle $v - av$ in which we will have a unitary current injection, we want to know how this will affect to the cycle $6 - 6'$. We will backtrack edge number 8 and edge number 6 while trying to equal the depth of $5'$ to the depth of vertex 5. This two edges will have a value of $J = -1$ each. Once we are in vertex 4, we will backtrack to reach the first common ancestor. Edge number 3 will be backtracked from the branch of the alias vertex and therefore will have an assigned value of $J = -1$. Edge 4 will be backtracked from the original vertex branch and will have an assigned value of $J = 1$. In the closed cycle of $5 - 5'$ we will have the sum of all the impedances. Now if we look at the cycle $6 - 6'$, what we have done only affects him in one edge, edge number 6, which is a common edge for both cycles. Therefore the voltage in the cycle $6 - 6'$ will only be affected by $Jz_{ue} = -z_6$.

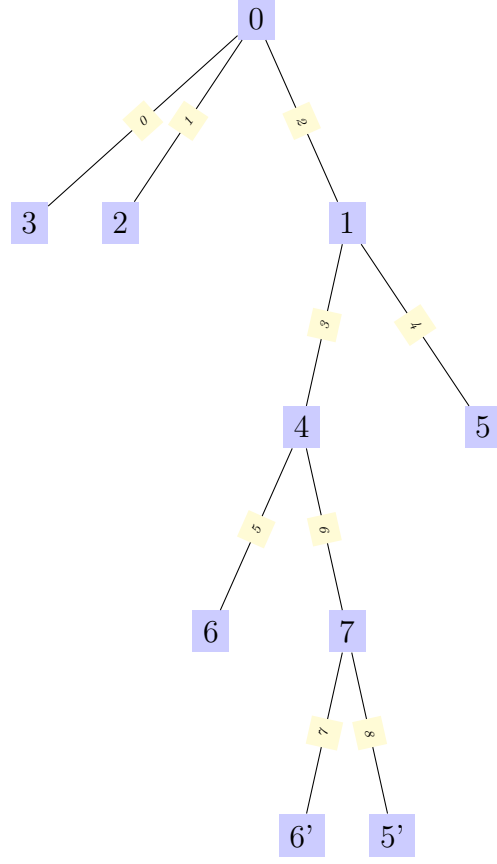


Figure 3.3.1: The original networks is at the right of this picture and the spanning tree at the left.

Example 3.3.1.

Notice that at the same time, this voltage drop will be the sum of the impedances on the common edges of the cycles $v - av$ and $u - au$. If we repeat what we have just explained for all the cycles we have in the graph, what we obtain is the sensitivity matrix.

The diagonal elements will be how the cycle is affected by the current in itself, the sum of all the impedances.

What we have is:

$$Z_{uc,vc} \begin{pmatrix} \Delta Q_{vc} \\ \Delta P_{vc} \end{pmatrix} = \begin{pmatrix} \Delta Er_{uc} \\ \Delta Ei_{uc} \end{pmatrix} \quad \text{where} \quad Z_{uc,vc} = \begin{pmatrix} X & R \\ -R & X \end{pmatrix}_{uc,vc},$$

where Er, Ei stand for real and imaginary part of complex voltages.

Notice that due to the hierarchical way in which we have structured the information, it is easy to perform upwards and downwards sweeps.

Now approaching the power flow problem again. Recall that we were trying to solve the system $M\Delta S = \Delta E$ for the given voltage values of the network. Since now we already know the sensitivity matrix, this can be solved by using the well-known Gauss elimination method with pivoting. Recall that we have a power S that is the sum of the consumed power S_{cons} and the compensated power S_{comp} : $S = S_{cons} + \Delta S_{comp}$.

And what we are iteratively correcting is ΔS_{comp} until voltage discrepancies completely disappear or the new corrected value is very small.

3.4 Losses in power of the system

The losses of active power may be known by a power balance of the active power injected to the network, taking into account the correct sign of the slack node 0 and of the other nodes:

$$L = \sum_{k \neq 0} P_k. \quad (3.4.1)$$

However, it can be predicted more accurately by using another formula that will contain less error. If we consider that the loss of active power happens in resistors, it can also be found by adding the active power lost in every branch and subtracting it to the total injected power:

$$L = \sum_l R_l J_l^2. \quad (3.4.2)$$

Chapter 4

Electrical Distribution Network Simulator

A simulator of the system was made in order to be able to easily interact with the program, and to allow a nice visualization of the network and the spanning tree that the program creates by means of using graphical interactive libraries. The OpenGL library was used to be able to show the figures of the networks. And the FLTK library was used to display several values on the screen and to integrate buttons on the simulator that allow changes on some controls [7], [3].

In figure 4 we see a screenshot of the simulator. In the middle we have the pictures of the structure of the network and the created spanning tree, and in both sides we have different controls and information of the outcome of the program. At the left-bottom corner all the general information of the network that the program provides is given such as: the header power, the total consumed power, the compensated power, and the total loss power.

This simulator provides a very comfortable environment that shows the voltage and power information of any desired knot just by clicking on it. The selected vertex turns green as well as its predecessor edge, furthermore we can see to which edge of the original network it corresponds because it also turns green. In figure 4 node 6' is selected. Additionally we can see the electrical information of this specific edge at the sides of the screen, particularly, the loss power in the edge, their impedance, the intensity, the voltage value, and the contracted power. There are also two buttons on this side of the screen the *previous*, and *next* which also allows us to select the previous branch as following the previously mentioned algorithms or the next branch. At the left of the screen there are some buttons that allow the users to increase or decrease the consumption factor at their will. There are two buttons for each direction we want the value to change. One that allows a small increase/decrease and one that allows a greater increase/decrease. Additional buttons to control the network can be added if desired, since having a simulator is a really good tool to take profit from and would allow more control of the network.

To summarize, this is a very intuitive simulator, that allows interaction with the program and it is the perfect tool to visualize the obtained results.

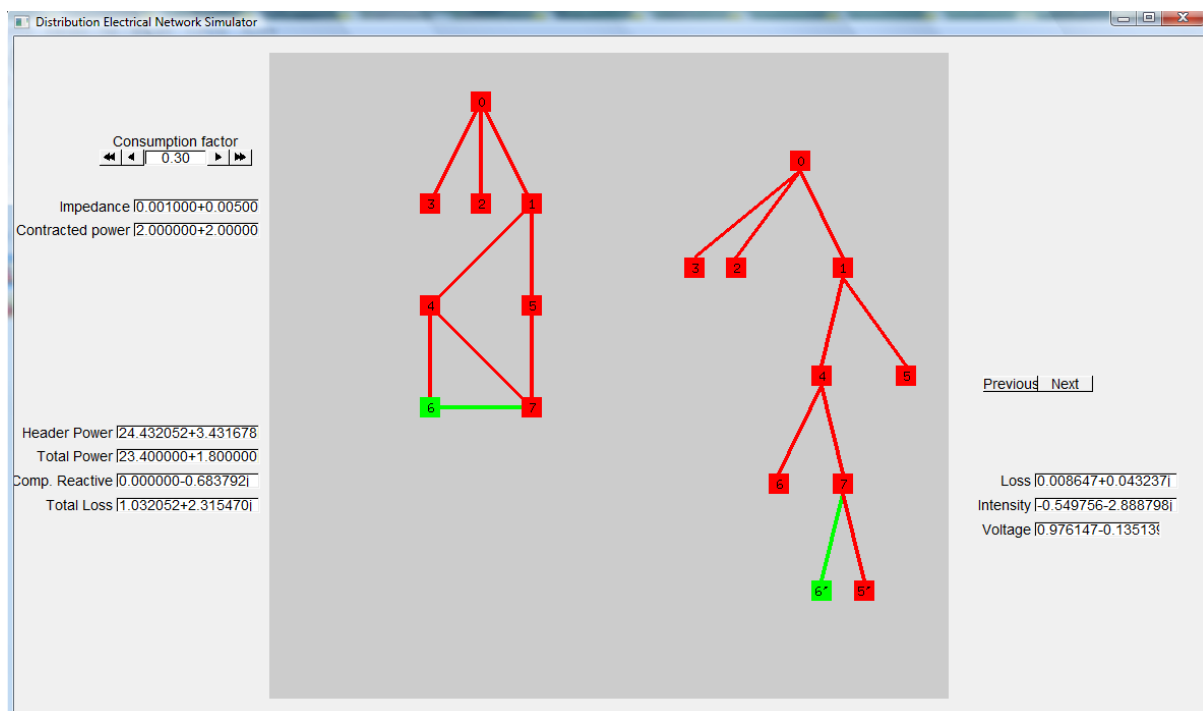


Figure 4.0.1: Electrical Distribution Network Simulator screenshot

Chapter 5

Results

In this chapter the outcome of the program is explained for a specific network that we have chosen to test the program and check its efficiency. We will also study how the program behaves when the consumption factor is increased in our particular network.

First of all, table 5.1 contains the nodes information that is being introduced to the program. Table 5.2 contains electrical information of the branches of the network, the information of the impedance is first given in the units of the program and later in units of the International System (SI). Furthermore, vertex number 7 is a PV node with a consigned voltage value of 25 kV.

Vertex	Active power (10MW)	Reactive power (10MVAR)
0	0	0
1	2	10
2	20	-8
3	10	20
4	10	2
5	14	20
6	2	2
7	20	-40

Table 5.1: Information of the nodal power

Edges (from- to)	Resistance	Reactance	Resistance (Ω)	Reactance (Ω)
0 (0-3)	0.005	0.001	3.125	0.625
1 (0-2)	0.001	0.003	0.625	1.875
2 (0-1)	0.001	0.005	0.625	3.125
3 (1-4)	0.005	0.005	3.125	3.125
4 (1-5)	0.002	0.004	1.250	2.500
5 (4-6)	0.003	0.001	1.875	0.625
6 (4-7)	0.001	0.005	0.625	3.125
7 (6-7)	0.001	0.005	0.625	3.125
8 (5-7)	0.001	0.005	0.625	3.125

Table 5.2: Edges information with the resistance and reactance in the units of the program and in Ω respectively

Vertex	Alias	Voltage	Current	Loss Power
0	0	(1, 0)	(11.9464, 1.42305)	
1	1	(0.99472, -0.0373123)	(7.3785, 0.419733)	(0.0546185, 0.273092)
2	2	(1.0005, -0.0102)	(3.01043, 1.16871)	(0.0104286, 0.0312857)
3	3	(0.989201, 0.0135)	(1.55747, -3.01149)	(0.0574741, 0.0114948)
4	4	(0.99118, -0.0624159)	(2.86439, 2.15634)	(0.0642725, 0.0642725)
5	5	(0.985303, -0.053951)	(4.26943, -0.219463)	(0.0365524, 0.0731048)
6	6	(0.990854, -0.0661106)	(0.46714, 1.07585)	(0.00412701, 0.00137567)
7	7	(0.997655, -0.0684371)	(0.908863, 1.47688)	(0.00300721, 0.0150361)
8	6	(0.990854, -0.0661106)	(-0.185824, -1.39739)	(0.00198722, 0.00993612)
9	5	(0.985303, -0.053951)	(-2.3107, -2.93254)	(0.0139391, 0.0696955)

Table 5.3: Voltage, current and loss power for a given network with $\mu = 0.15$.

Finally table 5.3 shows information with the outcome of the program in the units of the program. Notice that the voltage values of the alias vertex is the same as to their original vertex. Hence, the program has been able to properly solve the problem. The number of iterations for power compensation was 14, with an error smaller than 10^{-10} . To improve the functioning of the networks it is very interesting to know the total consumed power, the total injected power, the total lost power, and for the purposes of this project also the total compensated power. The obtained values for the given example are shown in table 5.4.

Total Consumed Power	(11.7, 0.9)
Total Lost Power	(0.246407, 0.549294)
Total Compensated Power	(0, -0.0262461)
Total Injected Power	(11.9464, 1.42305)

Table 5.4: Information that the program shows of the total power values

Notice that the real part of the total compensated power is zero, this is because the power is compensated between the pairs of alias and when add this term vanishes. The imaginary part of the total compensated power comes from the PV node, it comes from the correction in the reactive power of the vertex number 7 in our example. However, its voltage magnitude has not been modified.

Generally the consumption factor μ that a consumer has can be taken as a known parameter. Since between a margin it is predictable to know how much power will be consumed according to the contracted power they have established in their bills and how much they have been consuming in the past. Changes over time are usually slow so it is reasonable to take this as a constant value.

Using the simulator we can observe how the system behaves when the consumption factor is increased. Since the consumption factor has a linear relationship with the voltages, thus is linear with the power but it will be quadratic with the loss power. In fig. (5.0.1) we can see the plot of the consumption factor versus the loss power and we can observe the clearly quadratic behavior. When the consumption factor is highly increased the system may reach the voltage collapse where many assumptions cannot longer be made and for instance the steady state of the system is no longer achieved. This reveals that if suddenly all the consumers unpredictably start consuming more power at the same time,

the network would have problems. Another description of the system would be needed for this case.

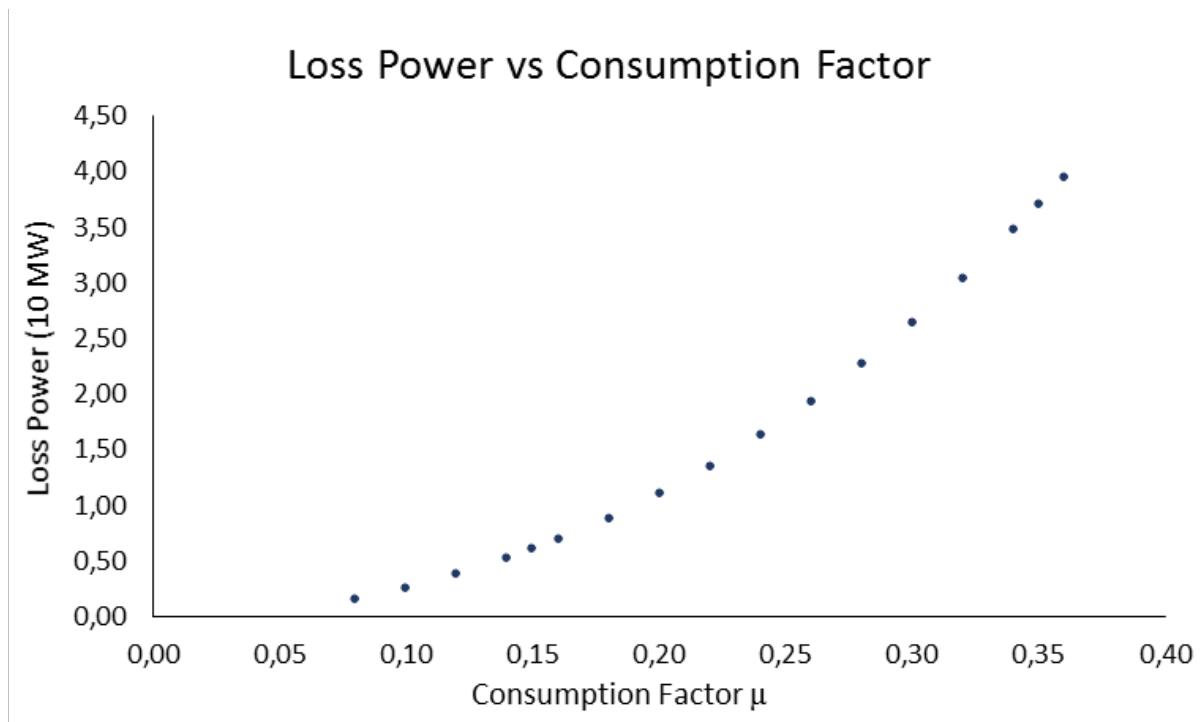


Figure 5.0.1: Loss power versus consumption factor in the given example

Conclusions

This project demonstrates the application of several numerical methods which are then used to arrive at solutions to the power flow problem. This study models and visualizes Electrical Distribution Networks using a *C++* code which stores data in hierarchical trees. One of the most important successes of this project is that this information can be stored in vectors thereby achieving a more efficient and time-saving solution.

The mathematical methods and calculations used in the *C++* code are based on knowledge gained from the courses taken in the undergraduate mathematics curriculum. These methods have been applied to developing meshed trees for modeling networks. The algorithms are written to traverse graphs in hierarchical patterns in forward and reverse directions. This allows the Shirmohammadi-Hong method to perform computations with the ability to correct electrical information in the tree iteratively. Structuring the information in an ordered fashion therefore is crucial to this project, which allows for a solution to the power flow problem in a more efficient way.

Although the efficiency of the method is high, in networks with higher impedances or higher loads the solutions accuracy or convergence may suffer due to large power losses in these networks which increase nonlinearly. Strongly nonlinear networks associated with higher impedances or power loads may produce non-convergence when the network is approaching voltage collapse. Additionally, the compensation methods described by [6] have not been tested in more highly meshed networks, which may limit their applicability in this project.

A simulator was created to interact with the networks to visualize their structures and to feed values back into the algorithm for iterative processing. The simulator is a very intuitive platform; it shows the network on screen with the built spanning tree as well as general information it contains. Furthermore, it allows for the selection of branches and gives their specific information. Consumption factor can be modified independently for predicting cutoff values for steady state operation.

Future work on this project could include adding more controls to the simulator in order to gain more insight into larger or more complex network. An ambitious improvement could be to allow modification of the structure of the network from the simulator by adding or removing branches to determine its effect on the whole network or another specific node. Additionally, one may be able to study how the power consumption changes over different periods of time, such as over the day and over the year. This requires a more accurate model of the habits of power consumption on different kinds of customers and their respective power loads. Finally, the minimization of power loss over a network could also be studied in future iterations, making this method to solve the power flow problem quite useful.

Bibliography

- [1] Bondy, J. A., and U. S. R. Murty. *Graph Theory*. New York: Springer, 2010. Print.
- [2] Evans, J. R., Minieka, E., (1992). *Optimization algorithms for networks and graphs*. New York: M. Dekker.
- [3] "Fast Light Toolkit." (n.d.) Retrieved June 27, 2016, from <http://www.fltk.org/index.php>
- [4] Gibbons, A. (1999). *Algorithmic graph theory*. Cambridge: Cambridge University Press.
- [5] Gonzalez, Fermin Barrero. *Sistemas de Energ'ia El'ectrica*. Madrid, España: Thomson, 2004. Print.
- [6] Luo, G., and Semlyen, A. (1990). *Efficient load flow for large weakly meshed networks*. IEE Trans. Power Syst. IEEE Transactions on Power Systems, Vol.5 No.4, pp.1309-1316. doi:10.1109/59.99382
- [7] The Industry's Foundation for High Performance Graphics (n.d.). Retrieved June 27, 2016 from <https://www.opengi.org/resources/libraries/glut/>
- [8] Shirmohammadi, D., Hong, H., Semlyen, A., and Luo, G. *A compensation-based power flow method for weakly meshed distribution and transmission networks*. IEEE Trans. Power Syst. IEEE Transactions on Power Systems, Vol.3, No.2, May 1988, pp. 753-762. doi:10.1109/59.192932